Homework 6 - Dictionaries and Try/Except Blocks

CS 1301 - Intro to Computing - Spring 2022

Important

- Due Date: **Tuesday March 1st, 11:59 PM**.
- This is an individual assignment. High-level collaboration is encouraged, **but your submission must be uniquely yours.**
- Resources:
  - TA Helpdesk
  - Email TA's or use class Piazza
  - How to Think Like a Computer Scientist
  - CS 1301 YouTube Channel
- Comment out or delete all function calls. Only import statements, global variables, and comments are okay to be outside of your functions.
- **Read the entire document before starting this assignment.**

The goal of this homework is for you to enhance your understanding of dictionaries and try/except blocks. The homework will consist of 5 functions for you to implement. You have been given the HW06.py skeleton file to fill out. However, below you will find more detailed information to complete your assignment. Read it thoroughly before you begin.

Hidden Test Cases: In an effort to encourage debugging and writing robust code, we will be including hidden test cases on Gradescope for some functions. You will not be able to see the input or output to these cases. Below is an example output from a failed hidden test case:

```
Test failed: False is not true
```

Written by Anastasiya Masalava (amasalava3@gatech.edu) & Andrew Hsu (yhsu72@gatech.edu)

Find Ticket

Function Name: findTicket()
Parameters: ticketDictionary (`dict`)
Returns: cheapestTicket (`tuple`)
Description: You and your Georgia Tech friends decided to see the Winter Olympic Games in Beijing, China. You want to find the cheapest airplane ticket for your trip. Given a ticket dictionary with names of airlines (`str`) as keys and prices (`int`) as values, write a function that returns a tuple with the name and price of the cheapest available option. If the ticket dictionary does not have any airline-price pairs, you should return `"No tickets available!"`

Note: No two flights will have the same ticket price, and no plane ticket price will exceed $10,000.

```
>>> ticketDictionary = {
        "Air Canada" : 722,
        "Alaska" : 931 ,
        "jetBlue": 871
        }
>>> findTicket(ticketDictionary)
('Air Canada',722)
```

```
>>> ticketDictionary = {
        "Air Seoul" : 700,
        "Asiana Airlines" : 500,
        "China Eastern" : 900,
        "Frontier" :1000
        }
>>> findTicket(ticketDictionary)
('Asiana Airlines', 500)
```

Find Hotel

Function Name: findHotel()
Parameters: hotelDictionary (`dict`)
Returns: preferredHotel (`dict`)
Description: Having found perfect airplane tickets for your trip, you now need to choose a hotel in Beijing that would satisfy you and your friends. Every friend votes for one hotel where they want to stay. Write a function that takes in one parameter: a dictionary that contains your friends' names (`str`) as keys and the hotels they chose (`str`) as values. Your task is to create a function that finds the most preferred hotel and returns a dictionary which contains the name of this hotel (`str`) as a key and the number of friends (`int`) who voted for this hotel as a value. There will be no hotels with the same number of votes. If the input dictionary does not have any friend-hotel pairs, you should return `"No hotels available!"`.

Note: You are encouraged to use .items() method in this function.

```
>>> hotelDictionary = {
        "Arvin" : "Grand Hyatt Beijing",
        "Craig" : "Grand Millennium Beijing",
        "Josh": "Grand Hyatt Beijing"
        }
>>> findHotel(hotelDictionary)
{"Grand Hyatt Beijing" : 2}
```

```
>>> hotelDictionary = {
        "Naomi" :  "Beijing Marriott Hotel",
        "Paige" : "Beijing Marriott Hotel",
        "Ramya" : "Beijing Marriott Hotel",
        "Cynthia" : "The Bulgari Hotel"
        }
>>> findHotel(hotelDictionary)
{"Beijing Marriott Hotel" : 3}
```

Find Event

Function Name: findEvent()
Parameters: myInterests (`list`), schedule (`dict`)
Returns: match (`list`)
Description: You are ready to watch some Olympic games. However, you are only interested in certain sports, so you are only going to the stadium on days with sports that interest you.
Given a list containing your sports interests and a dictionary that maps each date to a list of sport events, write a function that returns a list containing the dates that feature at least one of your interested sports. The list should not have repeated dates. If none of the dates contain any of the sports you are interested in, return an empty list.

Note: The returned list needs to be in **alphabetic order**.

```
>>> myInterest = ["Skiing","Snowboard","Curling"]
>>> schedule = {
    "Feb 18" : ["Skiing","Snowboard","Luge"],
    "Feb 19" : ["Skeleton", "Ski Jumping"],
    "Feb 20" : ["Skiing","Curling"]
            }
>>> findEvent(my_interest, schedule)
['Feb 18', 'Feb 20']
```

```
>>> myInterest = ["Luge","Ski Jumping","Figure Skating"]
>>> schedule = {
    "Feb 18" : ["Skiing","Snowboard","Luge"],
    "Feb 19" : ["Skeleton", "Ski Jumping"],
    "Feb 20" : ["Skiing","Curling"]
            }
>>> findEvent(my_interest, schedule)
['Feb 18', 'Feb 19']
```

Figure Skating

Function Name: figureSkating()
Parameters: technicalScores (`list`), componentScores (`list`)
Returns: finalScores (`list`)
Description: You and your friends decided to see a gorgeous figure skating competition. The event administrators displayed a technical score (`int`) and a program component score (`int`) for each figure skater separately. Write a function that takes in two lists: the list of technical scores for each competitor and the list of program component scores of each competitor. The function should return a list with final scores for each competitor. The final score for a competitor is calulated by the sum of their technical score and competitor score. However, some of the items in the given lists are not ints and will cause an error while adding them together so only include final scores that don't error in your list.
Note: A technical score in the first list corresponds to a program component score with the same index in the second list.
Note: You must use try/except in this function.

```
>>> technicalScores = [100, "&", 110, 130, 110]
>>> componentScores = ["*", 120, 110, 130, 130]
>>> figureSkating(technicalScores, componentScores)
[220, 260, 240]
```

```
>>> technicalScores = [180, ["Hi"], 150, 120, 100]
>>> componentScores = [80, 90, None, 120, 110]
>>> figureSkating(technicalScores, componentScores)
[260, 240, 210]
```

Sport Management

Function Name: sportManagement()
Parameters: countryDict (`dict`)
Returns: sportDict (`dict`).
Description: You are an organizer of the Winter Olympics, and your job is to figure out how what countries are participating in each sport category. Given a dictionary of countries' names mapped to the sports it participated in, write a function that returns a dictionary where the keys are sports and the values are lists of countries who participated in that sport.

Note: The list of contries returned should be in alphabetic order.

```
>>> countryDict = {"Belarus": ["Ice Hockey", "Skiing"],
                           "Lebanon": ["Skiing", "Snowboard"],
                           "Denmark":["Skiing", "Luge"]}
>>> sportManagement(countryDict)
{'Ice Hockey': ['Belarus'], 'Skiing': ['Belarus', 'Denmark','Lebanon'],
 'Snowboard': ['Lebanon'], 'Luge': ['Denmark']}
```

```
>>> countryDict = {'Japan': ['Skiing', 'Ice Hockey'],
                           'Taiwan': ['Skiing', 'Snowboard'],
                           'Romania': ['Bobsleigh', 'Luge', 'Skeleton']}
>>> sportManagement(countryDict)
{'Skiing': ['Japan', 'Taiwan'], 'Ice Hockey': ['Japan'],
 'Snowboard': ['Taiwan'], 'Bobsleigh': ['Romania'],
 'Luge': ['Romania'], 'Skeleton': ['Romania']}
```

Grading Rubric

Function	Points
findTicket()	20
findHotel()	20
findEvent()	20
figureSkating()	20
sportManagement()	20
Total	**100**

Provided

The `HW06.py` skeleton file has been provided to you. This is the file you will edit and implement. All instructions for what the functions should do are in this skeleton and this document.

Submission Process

For this homework, we will be using Gradescope for submissions and automatic grading. When you submit your `HW06.py` file to the appropriate assignment on Gradescope, the autograder will run automatically. The grade you see on Gradescope will be the grade you get, unless your grading TA sees signs of you trying to defeat the system in your code. You can re-submit this assignment an unlimited number of times until the deadline; just click the “Resubmit” button at the lower right-hand corner of Gradescope. You do not need to submit your `HW06.py` on Canvas.